public class Baby {
    private String name;
    private int age;

    public Baby(int age, String theName) {
        age = age;
        name = theName;
    }

    public String toString() {
        return name + " (age " + age + ")";
    }

    public static void main(String[] args) {
        Baby baby1 = new Baby(3, "Bill");
        System.out.println(baby1);
    }
}
Administrative Details

• Bill is gone! (and will be back at the end of next week)
  • His OH are canceled next week
  • I will take over the classes
  • Dan will take over the labs
• Lab1
  • Due on 2/11/18 (Sun) at 11pm
Agenda

• Inheritance
  ○ “extends”
  • abstract class
  • interface
  • “implements”
public class Cookie{
    private int calories;
    private boolean isExpired;

    public Cookie(int calories){
        this.calories = calories;
        isExpired = false;
    }

    public boolean isEdible(){
        return !isExpired;
    }

    public int getCalories(){
        return calories;
    }
}
public class CookieMonster{
    private int calories;
    public CookieMonster(){
        calories = 0;
    }
    public void eat(Cookie something){
        if(something.isEdible){
            int tempCalories = something.getCalories();
            calories += tempCalories;
            System.out.println("Me eat " + tempCalories +" calories! Om nom nom nom nom");
        }
    }
    public static void main(String[] args){
        ...
    }
}
public class Cookie{
    private int calories;
    private boolean isExpired;
    private int numChips;

    public Cookie(int calories){
        this.calories = calories;
    }

    public boolean isEdible(){
        return !isExpired;
    }

    public int getCalories(){
        return calories;
    }
}
public class ChocolateChipCookie extends Cookie {
    private int numChips;

    public ChocolateChipCookie(int numChips) {
        super(numChips * 2);
        this.numChips = numChips;
    }
}

By inheriting other classes, we can create specialized subclasses, e.g. “is a”

```java
public class ChocolateChipCookie extends Cookie{
    ... 
}
```

A subclass inherits all non-private fields and methods from the superclass

An object of a subclass type can be stored in a variable of a superclass type, e.g.

```java
Cookie chocoCookie = new ChocolateChipCookie(10);
```

Note, each class can **extend** up to 1 class
Agenda

- Inheritance
  - “extends”
  - abstract class
- interface
  - “implements”
Despite his voracious appetite for cookies, Cookie Monster shows awareness of healthy eating habits for young children and also enjoys fruits and eggplant.
public class Apple{
    private int calories;
    private boolean isChopped;
    
    public Apple(int calories){
        this.calories = calories;
        isChopped = false;
    }
    public boolean isEdible(){
        return isChopped;
    }
    public int getCalories(){
        return calories;
    }
    public void chop(){
        isChopped = true;
    }
}
public class Orange{
    private int calories;
    private boolean isPeeled;

    public Orange(int calories){
        this.calories = calories;
        isPeeled = false;
    }
    public boolean isEdible(){
        return isPeeled;
    }
    public int getCalories(){
        return calories;
    }
    public void peel(){
        isPeeled = true;
    }
}
Orange extends Apple?

- Orange inheriting Apple (or vice versa) is **not** appropriate.
  - Logically, Orange “is not an” Apple
  - More practically, we don’t want isChopped nor chop() in Orange

- **Solution:**
  - Create an abstract class Fruit that implements common functionality (methods)
    - if the given method’s implementation is different, declare the method without a body
  - Have Apple and Orange inherit Fruit
public abstract class Fruit{
    private int calories;

    public Fruit(int calories){
        this.calories = calories;
    }

    abstract public boolean isEdible();

    public int getCalories(){
        return calories;
    }
}

public class Orange{
    private int calories;
    private boolean isPeeled;

    public Orange(int calories){
        this.calories = calories;
        isPeeled = false;
    }
    public boolean isEdible(){
        return isPeeled;
    }
    public int getCalories(){
        return calories;
    }
    public void peel(){
        isPeeled = true;
    }
}

public class Apple{
    private int calories;
    private boolean isChopped;

    public Apple(int calories){
        this.calories = calories;
        isChopped = false;
    }
    public boolean isEdible(){
        return isChopped;
    }
    public int getCalories(){
        return calories;
    }
    public void chop(){
        isChopped = true;
    }
}
public class Apple extends Fruit{
    private boolean isChopped;

double Apple(int calories){
    super(calories);
    isChopped = false;
}

public boolean isEdible(){
    return isChopped;
}

public void chop(){
    isChopped = true;
}
}
public class Orange extends Fruit{
    private boolean isPeeled;

    public Orange(int calories){
        super(calories);
        isPeeled = false;
    }

    public boolean isEdible(){
        return isPeeled;
    }

    public void peel(){
        isPeeled = true;
    }
}
Abstract Class

• An abstract class allows for a *partial* implementation
  • It contains at least 1 abstract method
    ```java
    public abstract class Fruit{
        ...
        abstract public boolean isEdible();
        ...
    }
    ```

  *Can’t be instantiated, i.e.,
  ```java
  new Fruit(); // throws an error
  ```
Agenda

- Inheritance
  - “extends”
  - abstract class
  - interface
  - “implements”
Cookie Monster wants to eat fruits!

```java
public class CookieMonster{
    private int calories;
    public CookieMonster(){
        calories = 0;
    }
    public void eat(Cookie something){
        // I can only eat cookies!
        if(something.isEdible()){
            int tempCalories = something.getCalories();
            calories += tempCalories;
            System.out.println("Me eat " + tempCalories
                                +" calories! Om nom nom nom");
        }
    }
    public static void main(String[] args){
        ...
    }
}
```
public void eat(Cookie something) {
    if (something.isEdible()) {
        int tempCalories = something.getCalories();
        calories += tempCalories;
        System.out.println("Me eat " + tempCalories + " calories! Om nom nom nom nom");
    }
}

public void eat(Cookie Fruit something) {
    if (something.isEdible()) {
        int tempCalories = something.getCalories();
        calories += tempCalories;
        System.out.println("Me eat " + tempCalories + " calories! Om nom nom nom nom");
    }
}
Interface

• Interface simply declares methods

```java
public interface Edible{
    public boolean isEdible();
    public int getCalories();
}
```

• When a class “implements” an interface, those methods are implemented in the class, e.g., Cookie implements isEdible() and getCalories()

```java
public class Cookie implements Edible{
}
```

• An object of a class type can be stored in a variable of an interface that the class “implements”, e.g.,

```java
Edible chocoCookie = new Cookie(10);
```

• Note, a class can **implement** many interfaces.
public void eat(Cookie something){
    if(something.isEdible()){
        int tempCalories = something.getCalories();
        calories += tempCalories;
        System.out.println("Me eat " + tempCalories
             +" calories! Om nom nom nom nom");
    }
}

public void eat(Fruit something){
    if(something.isEdible()){
        int tempCalories = something.getCalories();
        calories += tempCalories;
        System.out.println("Me eat " + tempCalories
             +" calories! Om nom nom nom nom");
    }
}
Example Class Hierarchy

- Edible
  - Fruit
    - Apple
    - Orange
  - Cookie
    - ChocolateChipCookie

Interface
Abstract Class
Class
public class Baby {
    ...
    public String toString() {
        return name + "(age " + age + ")";
    }
    ...
}
public class BossBaby extends Baby {
    ...
    public String toString() {
        return name + "(age " + age + ", CEO)";
    }
    ...
}
public class Driver {
    public static void main(String[] args) {
        Baby baby = new BossBaby(3, "Bill");
        System.out.println(baby);
    }
    ...
}